HL7Fuse

Developer guide, V1.0

Division by Zero

Table of Contents

Introduction....3
1 How Rosi works....4
1.1 General technical aspects....4
1.2 The overall process....4
2 Integrating with Rosi....6
2.1 Connecting directly to our services....6
2.1.1 REST services....6
2.1.2 JSON....6
2.1.3 Available Rosi services....7
2.2 .Net....9
2.2.1 Requirements....9
2.2.2 Example....10
2.3 Java....10
2.3.1 Requirements....11
2.3.2 Example....11
2.4 PHP....12
2.4.1 Requirements....12
2.4.2 Example....12
2.5 Other development languages....12
2.5.1 Requirements....12
3 Testing your application....13
4 Random password generator....14

Introduction

On my blog I get a lot of questions on how to set up a complete .Net system for HL7 message integration. In other words: all over the world developers create integration components from scratch to add HL7 integration to their applications. After working for a while with NHapi, the most complete and free component to support HL7 with .Net, I started to miss functionality. To make my life easier (and hopefully the life of other developers, I created the NHapiTools.

After that I build HL7Fuse. HL7Fuse isn't easy to describe in one word or sentence. It is based on SuperSocket and implements the MLLP/HL7 protocol, including SSL/TLS support. So using HL7Fuse you have a complete protocol implementation and you can focus on building the business logic for your application. Without any business logic implementation HL7Fuse can be used to act as a test HL7 server for your development environment.

Added to this HL7Fuse provides a standard implementation of such a business logic component, called the hub. The Hub allows you to receive HL7 messages and forward these messages based on routing rules. The message will be forwarded to an endpoint. A few standards endpoints, like a file end point or a MLLP/TCP end point, are also provided by the Hub. Of course, using the Endpoint interface you can always add your own endpoints.

I'd like to describe HL7Fuse as a Swiss pocket knife for your .Net/HL7 development needs. If you have any questions, please contact me through my blog at http://www.dib0.nl.

The sources and releases of NHapiTools and HL7Fuse can be found on Github:

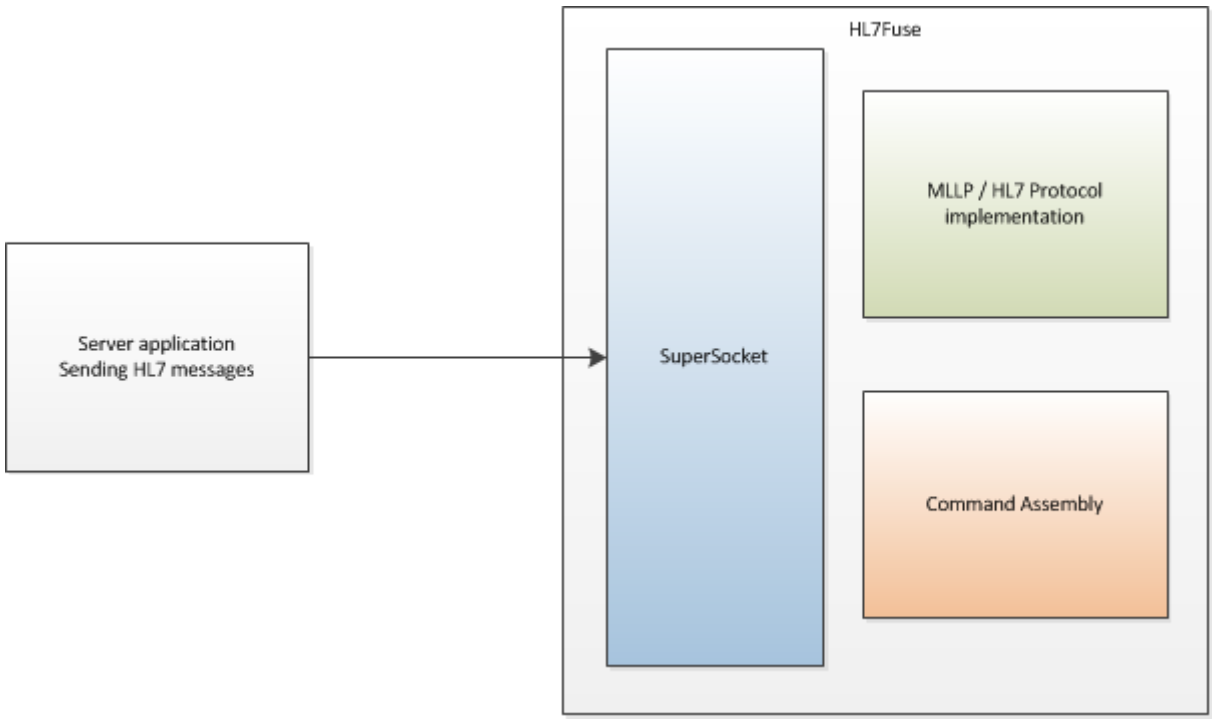
https://github.com/dib0/NHapiTools

https://github.com/dib0/HL7Fuse

Note: A basic understanding of HL7 and NHapi is assumed in this document.

1 How HL7Fuse works

HL7Fuse is based on SuperSocket (http://docs.supersocket.net/). There is quite a lot of documentation on this product, so I won't be going in to detail on SuperSocket and stick to HL7Fuse.



HL7Fuse in it's basic form is a configured SuperSocket application with a HL7 (using MLLP) over TCP/IP. SuperSockets provides the possibility to add SSL or TLS as encyption layer to the TCP/IP connection.

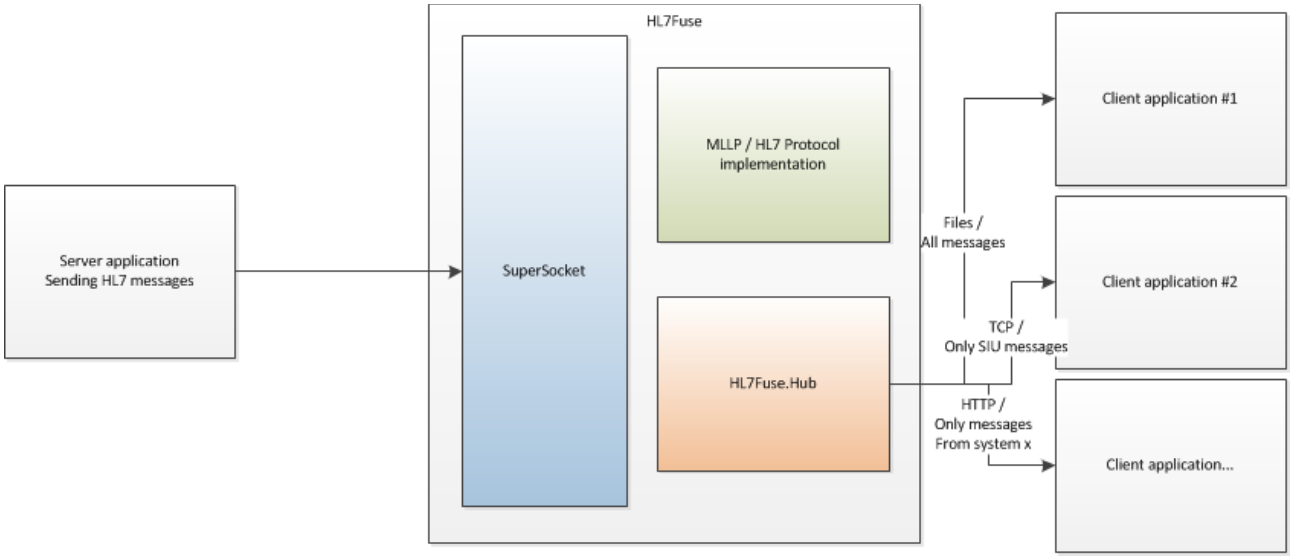
So the server application will send a HL7 message, which is received by HL7Fuse. HL7Fuse will cover all the protocol details for you. After receiving the message HL7Fuse will parse the message using NHapi. After successfully parsing the message to a NHapi Imessage object, SuperSocket will call the, so called, command assembly for further processing. After the processing is done, HL7Fuse will automatically return a ACK message to the server application. This can be a AA or AE message, based on configuration, implemented HL7 events and error states from the command assembly.

Without a command assembly HL7Fuse is just an implementation of the HL7 protocol over TCP/IP (using MLLP). And can be used as a development server that consumes/parses the HL7 message and returns an ACK.
You can implement your own command assembly to implement the logic needed for HL7 integration on your application.

1.1 HL7Fuse.Hub

The HL7Fuse.Hub assembly is an implementation of a command assembly. This implementation provides several endpoints and rule based routing of HL7 messages to these endpoints.

In other words, using the HL7Fuse.Hub command assembly, you have a HL7 message broker that allows you to receive messages and forward them to various endpoints using various protocols. Also you are able to manipulate the messages using your own implementation of the IMessageHandler interface.



2 The basic configuration options